

Practice Session 4: Working with Kubernetes

Make sure that you have already gone through [Lab-03](#)

The aim of this practice session is to make you acquainted with Kubernetes. Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation [1]. *Google* originally designed Kubernetes, but the Cloud Native Computing Foundation now maintains the project [1].

In this practice session, you are going to deploy the Kubernetes cluster on the ETAIS private cloud. You will also learn to orchestrate and deploy application services using Kubernetes CLI known as `kubectl` and the Kubernetes dashboard (the web UI alternative to CLI). Kubernetes container deployment has several advantages such as high availability, scalability, and faster CI/CD service deployment in the DevOps ecosystem.

References

Referred documents and websites contain supportive information for the practice.

Manuals

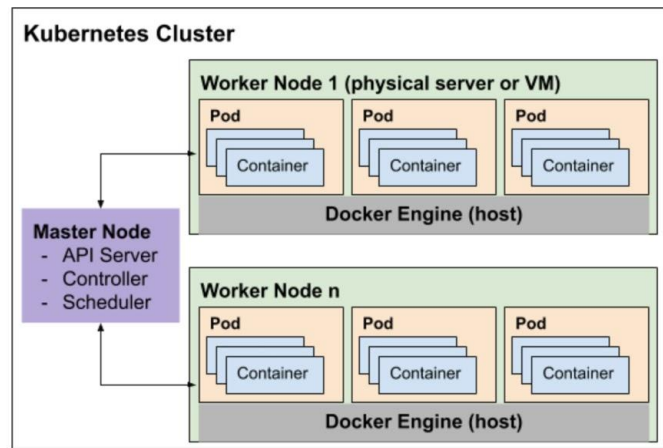
1. [What is Kubernetes](#)
2. [How does Kubernetes work](#)
3. [Kubernetes components](#)
4. [A short demo](#)

Introduction to Kubernetes

[Kubernetes](#) is an open-source container orchestration platform that automates the biggest part of the manual processes involved in deploying, managing and scaling containerized applications [1].

Kubernetes provides you with a platform for scheduling and running containers in clusters of physical or virtual machines. The Kubernetes architecture divides the cluster into components that work together to maintain the defined state of the cluster.

The Kubernetes cluster is a set of node machines for running container applications. The clusters can be understood as two parts: a management layer and computing machines or nodes. Each node has its environment and can be either a physical or a virtual machine. Each node runs pods consisting of containers.



Kubespray

Kubespray is a powerful open source tool to configure, deploy and effortlessly manage the Kubernetes cluster on various infrastructure platforms such as public clouds or bare metal servers. It uses ansible and kubeadm to install and set up the cluster.

Getting Started

- Make sure that you have a “**controller**” VM in your project.
- The “**controller**” VM should be using **m3.tiny** (2 vCPUs + 2GB RAM) flavor and **15GB** Storage.
- Rename the VM name from “**controller**” -> “**k8s-controller**”
 - Open your project from ETAIS profile
 - Go to “resources” -> “VMs”
 - Find the “Edit” option from the “Actions” list.
- Please go through the Lecture [slides](#)
- Open ETAIS profile (<https://minu.etais.ee/profile/>)
- Open your Project
- Go to “Resources” -> “Private clouds”
- Find “Change limits” from the “Actions” list.
- Increase the limit as follows:
 - Cores: from 4 to 14
 - RAM: from 8 to 26
 - Storage: from 20 to 60

Exercise 1. Installing Kubernetes using Kubespray.

There are several ways to install Kubernetes atop your resources. However, in this practice session, we will use Kubespray, which uses the *Ansible* automation tool. At this point, Ansible is expected to be new for you. We will cover Ansible (broadly on Automation topic) in a later lecture.

In this task, you will configure the Kubernetes nodes using Ansible on *CentOS 9*.

Task 1.1: Setting up Virtual Machines for K8s Cluster

- You need **four** virtual machines with the names k8s-controller (You can reuse the VM from Practice Session 2), k8s-master, k8s-worker1, k8s-worker2. Create the VMs with the following configurations:
 - image: *Centos 9*,
 - Flavor: *m3.xsmall (4 vCPU, 8 GB RAM)*
 - Storage: *15 GB*
 - Security Group: `allow-all`
 - Auto assign IP
- Log in to k8s-controller VM and install git, pip packages
 - Install git (If not present): `sudo yum install git`
 - Install pip (If not present): `python -m ensurepip --upgrade && python -m pip install --upgrade pip`
- Copy the ssh key from your local machine to `~/ .ssh` directory of *k8s-controller* machine. You may use `scp` command.
- Change the permissions after copying the ssh key: `chmod 400 ~/.ssh/ssh_key`

Task 1.2: Setting up of K8s Cluster

- Download *kubespray* repository and set up the configuration file to install the Kubernetes cluster **from** the **controller** node.
 - Make sure that you are in the k8s-controller VM.
 - Go to the `$HOME` directory. (`/home/<username>`):


```
cd ~
```
 - Download the package:
 - `git clone https://github.com/kubernetes-sigs/kubespray.git`
 - Install required pip packages: `cd kubespray && pip install -r requirements.txt`
 - Copy the inventory files : `cp -rfp inventory/sample inventory/mycluster`
 - Declare node IPs added to the cluster:
 - `declare -a IPS=(k8s-master k8s-worker1 k8s-worker2)`
 - Make sure to use Internal IP
 - Ex: `declare -a IPS=(192.168.42.159 192.168.42.68 192.168.42.191)`
 - Set Configure file with IPs:
 - `CONFIG_FILE=inventory/mycluster/hosts.yaml python3 contrib/inventory_builder/inventory.py ${IPS[@]}`

- Run the ansible playbook to install kubernetes:
 - `ansible-playbook -i inventory/mycluster/hosts.yaml --become --become-user=root cluster.yml --private-key ~/.ssh/<YOUR_SSH_KEY>`
- Login to master node (node1) and use the `kubectl` to work with the cluster
 - List the nodes in the cluster : `kubectl get nodes`
 - If you get a kubectl error message (!Connection refused), please follow the steps to configure the kube config file
 - `mkdir -p $HOME/.kube`
 - `sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config`
 - `sudo chown $(id -u):$(id -g) $HOME/.kube/config`

Exercise 2. Working with the Kubernetes cluster.

The exercise aims to work with the k8s cluster and get acquainted with kubectl commands to create pods, deployments, and services.

Task 2.1: Working with Pods

[Pods](#) are the smallest, most basic deployable objects in Kubernetes. They can hold one or a group of containers, such as Docker containers which share the storage, network, and other resources.

- Complete task 12.2.1 from here (<https://courses.cs.ut.ee/2022/cloud/spring/Main/Practice12>) to get acquainted with kubectl to manage pods.

Task 2.2: Working with deployments

Deployments describe the desired state of Kubernetes. They dictate how Pods are created, deployed, and replicated. Deployment files are written in YAML. YAML is a superset of JSON, which means that any valid JSON file is also a valid YAML file. Here is the [link](#) where you can find additional information about YAML and how it is used in Kubernetes.

- In the master node (node1), create a deployment file with the name `firstdeployment.yaml`
- Paste the following code (*Be careful with copying these blocks. It requires spaces instead of tabs*):

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```

name: nginx-deployment
labels:
  app: mynginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mynginx
  template:
    metadata:
      labels:
        app: mynginx
    spec:
      containers:
        - name: mynginx
          image: nginx
          ports:
            - containerPort: 80

```

- Run the deployment: `kubectl apply -f firstdeployment.yaml`
- Check if deployments were created: `kubectl get deploy`
- Check which pods were brought up: `kubectl describe deploy`
- List the pods: `kubectl get pods`
- Get the additional info about pods: `kubectl describe pods`
- Using the command from the last step, get an IP of pods and curl it from inside the VM. You should be able to see the webpage: `curl <IP-address-of-a-pod> .`
- Try accessing the Nginx pod in your browser (`http://node1_IP:80`). You should see “Unable to connect”

Task 2.3: Working with Services

Service enables network access to a set of Pods in Kubernetes. Services select Pods based on their labels. In the previous task, the pod is only accessible within the cluster. A service is an object that makes a collection of Pods with appropriate labels available outside the cluster and provides a network load balancer to distribute the load evenly between the Pods.

- Create a service to access the Nginx server outside the VM. Modify the `firstdeployment.yaml` by adding service. Append the following code,

```
---
```

```

apiVersion: v1
kind: Service
metadata:
  name: service-nginx
spec:
  type: NodePort
  selector:
    app: mynginx
  ports:
    - protocol: TCP
      port: 80 # internal cluster port
      targetPort: 80 # Port your application listen on in pods/containers
      name: tcp-80

```

- Create the deployment as an earlier step.
- List the service created: `kubectl get svc` and note down the nodeport assigned to this service. For example, nodeport_address: 31899
- Now open the Nginx in the browser using node1 IP : `http://node1_IP:<nodeport_address>`

Screenshot - 1

- Take a screenshot of a webpage where IP address are clearly seen.

Exercise 3. Deploy a flask application

Deploy your flask application using the docker image built during Practice session 2- Exercise 4. This exercise can be carried out by yourself. (Make sure, your in k8s-master VM)

- Create a deployment file with the name `flask-deployment.yaml` with `kind: Deployment`
 - This is similar to `firstdeployment.yaml`
 - Mention your docker image name with flask application image pushed to docker hub in your previous [practical session 2- Exercise 4](#)
 - Container Port should be 5000
- Create a service `kind: Service` to access the application outside Kubernetes network using NodePort (This is similar to Task 2.3. Here, port and targetPort would be 5000).
- Download the CO2.csv to `/mnt/data` (You can create `data` directory under `/mnt`) directory location in node1 VM: `wget https://gitlab.cs.ut.ee/poojara/k8s-cicd/-/raw/main/app/CO2.csv`
- You have to use [Persistent Volume](#) to store the `CO2.csv` file. To make access of `CO2.csv` file to flask application pod, you have create - Persistent Volume, Persistent volume claim
 - To create Persistent Volume volume, create a file with name `pv.yaml`

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pvolume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /mnt/data
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - node1

```

- To create Persistent Volume Claim (PVC) for the flask application pod, append the following block in to the deployment file `flask-deployment.yaml`

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: vclaim
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: manual

```

- Add the following block in flask-deploymet.yaml in the containers specification. Here, your mounting “/data ” directory to the persistent volume by using persistent volume claim.

```

lifecycle:
  postStart:
    exec:
      command: ["cp", "/data/CO2.csv", "/app/CO2.csv"]
  volumeMounts:
    - mountPath: "/data"

```

```

    name: data-storage-volume
  volumes:
    - name: data-storage-volume
      persistentVolumeClaim:
        claimName: vclaim

```

- Deploy the pv.yaml and deployment file using `kubectl`
- Check for the status of the pod `kubectl get pod -o wide`
- List the created persistent volume, volume claim
- Open the flask application in your browser (Check for the nodeport assigned to your flask application by service you created)

Screenshot - 2

- Take a screenshot of a webpage where your name and IP address are clearly visible.

Exercise 4. Working Kubernetes and Gitlab

In this exercise, you are going to deploy applications via GitLab CI/CD. To perform this task, the following need to be configured:

1. [Gitlab runner](#): It's an application that works with GitLab CI/CD to run jobs in a pipeline.
2. [Gitlab Kubernetes agent](#): To connect a Kubernetes cluster to GitLab, we must install an agent in our cluster.

Task 4.1: Installing [helm](#)

Helm is the package manager for Kubernetes. Here, we are using helm to install GitLab runner and k8s agent for GitLab.

- Make sure that you're in k8s-master VM.
- Download helm repository: `wget https://get.helm.sh/helm-v3.9.2-linux-amd64.tar.gz`
- Extract the files: `tar -zxvf helm-v3.9.2-linux-amd64.tar.gz`
- Setup helm: `sudo mv linux-amd64/helm /usr/local/bin/helm`
- Test the installation: `helm help`

Task 4.2: Create gitlab project and setup k8s agent for gitlab

- Create a project in the gitlab with name `k8s-deployment` under group `Devops2022fall/students/devops2022Fall-<lastname>-<studyCode>`
- Add the `flask-deployment.yaml` and `pv.yaml` files into the gitlab project.
- Setup k8s agent for your gitlab project:

- Move to k8s-deployment gitlab project.
- Go to the Infrastructure tab from the left menu of your gitlab project.
 - Choose Kubernetes Cluster
 - Click on Connect to Cluster
 - Click on the drop down menu, enter **agent** on the search box and Click on create **agent:agent** and then click on **register**.
 - After the previous step, you see instructions to install using helm and token details as shown below

Install using Helm (recommended)

From a terminal, connect to your cluster and run this command. The token is included in the command.

```
helm repo add gitlab https://charts.gitlab.io
helm repo update
helm upgrade --install agent gitlab/gitlab-agent \
  --namespace gitlab-agent \
  --create-namespace \
  --set image.tag=v15.2.0 \
  --set config.token=MRfZQX8zfJgPT6fox3X1jQGwbW8XUz2PsfZxiAud-8HB2yoeiw \
  --set config.kasAddress=wss://gitlab.cs.ut.ee/-/kubernetes-agent/
```

- Copy and run all the helm commands in your k8s-master VM.
- After successful in creating agent, you should see on the gitlab

Agent				
Name	Connection status	Last contact	Version	Configuration
k8s-agent	Connected	5 minutes ago	15.2.0	Default configuration

- You can also check the agent installation on the k8s master node, `kubectl get po -n gitlab-agent`

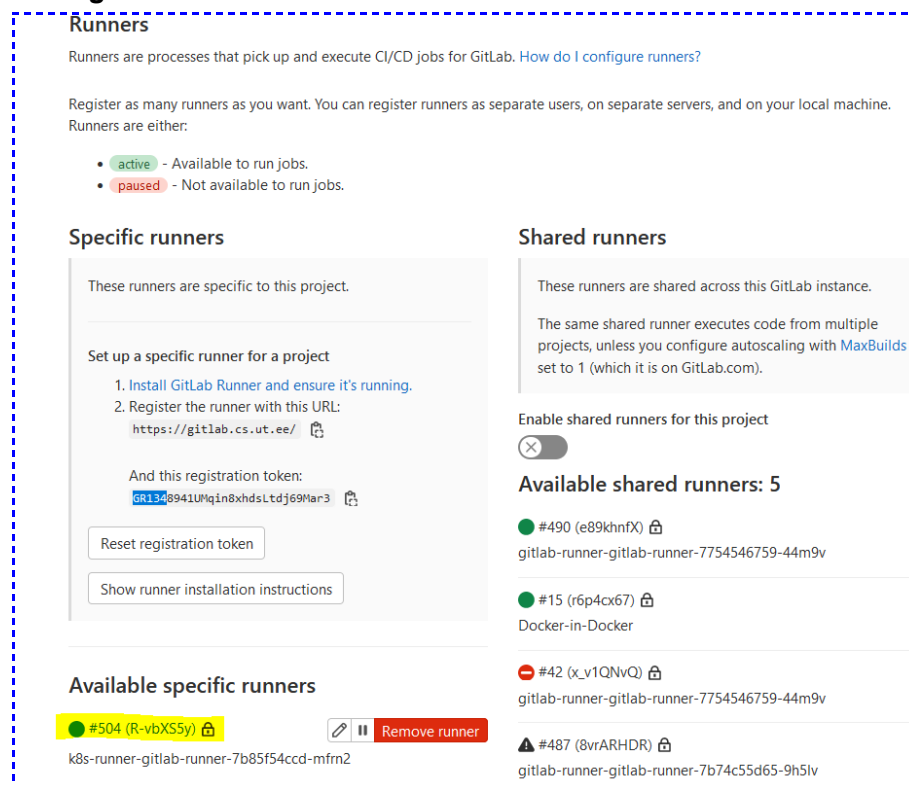
Task 4.3: Setup of gitlab runner for the project

Gitlab runners are used to run the jobs in the pipeline in the designated infrastructure. You can choose the various executors for your runner for example, Shell, Docker etc.

- Go to the Settings of the gitlab project and Click on **CI/CD**.
- Move to **Runners** and click on **Expand** button and note down **URL** and **registration token**.
- In master node, install the gitlab runner using helm as mentioned below.
 - Add gitlab charts to your helm repo: `helm repo add gitlab https://charts.gitlab.io`
 - Initialise the helm : `helm init`

- Download the values.yml, this is the configuration file required to install the runner: `wget https://gitlab.com/gitlab-org/charts/gitlab-runner/-/raw/main/values.yml`
- Modify the values.yml to add your project settings (Use your URL and Registration token noted in the previous step) :
 - Search for `gitlabUrl` and uncomment it, modify to `gitlabUrl`: `http://gitlab.your-domain.com/` for example `gitlabUrl: https://gitlab.cs.ut.ee/`
 - Search for `runnerRegistrationToken` and modify to `runnerRegistrationToken: ""`, for example `runnerRegistrationToken: "GR134xxxxxx"`
- Install the runner using helm : `helm install --namespace default gitlab-runner -f values.yml --set rbac.create=true gitlab/gitlab-runner`
- Note: If you made a mistake and couldn't find a registered runner, but already installed a helm chart, you might need to delete your helm chart and do the previous step carefully. Use `helm list` to list all charts and find the correct release. Then delete it by `helm delete <release-name>` EX: `helm delete gitlab-runner (src: https://www.ibm.com/docs/en/imdm/11.6?topic=helm-deleting-chart)`
- After successful installation, you should see a registered runner here.

Settings→CI/CD→Runners



Screenshot - 3

- Take a screenshot of a command output `kubectl get po -o wide`.

Task 4.4: Deploying gitlab project jobs on k8s infrastructure.

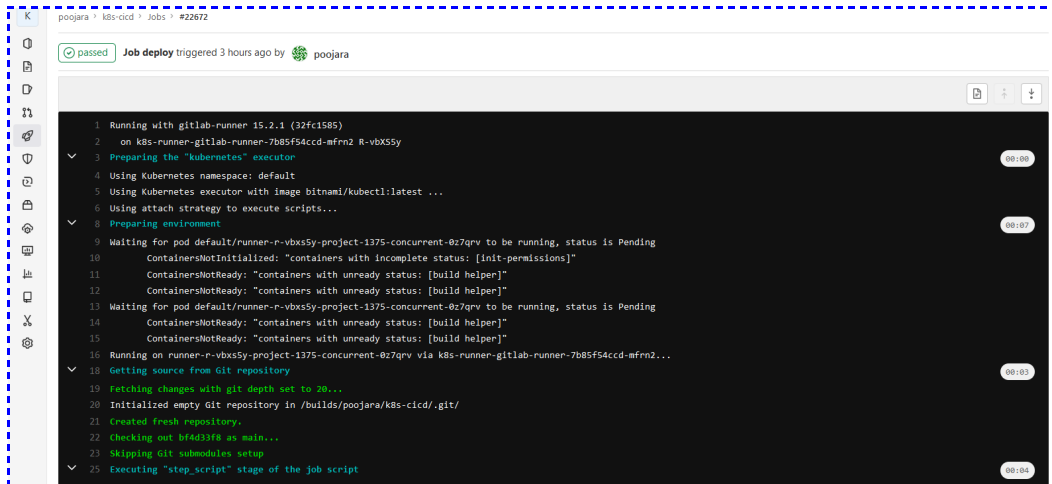
In this task, you're going to create a **.gitlab-ci.yml** file containing the jobs to deploy on k8s cluster.

- Go to gitlab project and create a file **.gitlab-ci.yml**
- Add the following content to the file and change the “*kubectl config use-context poojara/shiva-k8s-deployment:agent*” to *kubectl config use-context <your_project_path>/k8s-deployment:agent*

```
stages:
  - deploy

deploy:
  stage: deploy
  image:
    name: bitnami/kubectl:latest
    entrypoint: [""]
  script:
    - kubectl config get-contexts
    - kubectl config use-context poojara/shiva-k8s-deployment:agent
    - kubectl config current-context
    - kubectl get pods
    - kubectl delete -f ./flask-deployment.yaml || true
    - kubectl delete -f ./pv.yaml || true
    - kubectl apply -f ./pv.yaml
    - kubectl apply -f ./flask-deployment.yaml
```

- Define the stages, here deploy (which deploys the job to k8s cluster)
- Under deploy job:
 - image with name `bitnami/kubectl:latest` used to run the commands mentioned in the script.
 - Script section contains the `kubectl` commands to setup kube configurations using agent and create the deployments on the k8s cluster.
 - Make sure that, you updated *kubectl config use-context*
- After committing the code, you should see the pipeline execution status.
 - Go to **CI/CD** → **Pipelines** tab on the left side of your project.
 - Click on the latest pipeline and visualise the execution order of your scripts like as shown below



- Go to the master node and check the for the created deployments
 - Check for the flask application deployment: `kubectl get deployments, kubectl get po -o wide`
 - If status of the pod is running, than deployment is successful and see the application running
 - Note down the port number assigned using : `kubectl get svc`
 - In your browser type, `http://node1_IP:<nodeport_address>`
 - You should see the flask application with CO2 values.
 - If status of the pod is CrashLoopBackOff, than check the logs of the pod using `kubectl logs <pod_name>`

Deliverables

1- Gather all the screenshots

- [Screenshot 1](#)
- [Screenshot 2](#)
- [Screenshot 3](#)

2- zip the deployment, service and persistent volume yaml files from Exercise 2 and 3 and all the screenshots.

3- Upload the zip file to the course wiki page.

4- You may **Stop** the Virtual Machines and you can start using the same in the next **practice session**.

Don't delete your VMs


References:

[1]: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/> , Accessed online : 11 May 2022

Some more resources

- This material (below link) contains material about installing and handling the kubernetes using Rancher.
<https://courses.cs.ut.ee/2021/DevOps/fall/Main/Lab03>

Troubleshoot

 This job is stuck because the project doesn't have any runners online assigned to it.
Go to project [CI settings](#)

This could be because *no available runners* are free where the pipeline can be assigned. Maybe runners are busy with other jobs.

Solution:

You can install an executor on your own machine (e.g. k8-controller VM) with following parameters:

- **Register the runner with this URL:** <https://gitlab.cs.ut.ee/>
- **Registration token:** your token
- Enter your **description**
- Enter a **tag**: Let's say k8-app-deployer. This will be used to select this runner while submitting the pipeline and running all the jobs in this runner.
- Enter an optional maintenance note. This is optional.
- Enter the **executor** : [Docker](#)
- **Image for the executor** : [bitnami/kubectl:latest](#)

Sample Screenshot:

```
[centos@k8-controller ~]$ sudo gitlab-runner register
Runtime platform                                arch=amd64 os=linux pid=1517
34 revision=76984217 version=15.1.0
Running in system-mode.

Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.cs.ut.ee/
Enter the registration token:
GR1348941ptvLYy6MNC3iZw1tnAw-
Enter a description for the runner:
[k8-controller.novalocal]: docker in docker for application deployment in kubern
ets luster.
Enter tags for the runner (comma-separated):
k8-app-deployer
Enter optional maintenance note for the runner:

Registering runner... succeeded                  runner=GR1348941ptvLYy6M
Enter an executor: docker+machine, docker-ssh, shell, virtualbox, ssh, docker-ss
h+machine, kubernetes, custom, docker, parallels:
docker
Enter the default Docker image (for example, ruby:2.7):
bitnami/kubectl:latest
Runner registered successfully. Feel free to start it, but if it's running alrea
dy the config should be automatically reloaded!
[centos@k8-controller ~]$
```